

Exhibit C

UDDI Programmer's API 1.0 UDDI Published Specification, 28 June 2002

This version:

<http://www.uddi.org/pubs/ProgrammersAPI-V1.00-Published-20020628.pdf>

Latest version:

<http://www.uddi.org/pubs/ProgrammersAPI-V1.00-Published-20020628.pdf>

Authors (alphabetically):

Toufic Boubetz, IBM
Maryann Hondo, IBM
Chris Kurt, Microsoft
Jared Rodriguez, Ariba
Daniel Rogers, Microsoft

Copyright© 2000-2002 by Ariba Inc., International Business Machines Corporation, Microsoft Corporation. All Rights reserved.

Contents

CONTENTS.....	2
INTRODUCTION.....	5
Document Overview.....	5
What is this UDDI anyway?	5
Compatible registries.....	5
What are tModels?.....	6
Classification and Identification information.....	7
DESIGN & ARCHITECTURE.....	8
Design Principles.....	8
Security.....	8
Versioning.....	8
SOAP Messaging.....	9
XML conventions.....	9
Error Handling.....	9
White Space.....	9
XML Encoding.....	9
API REFERENCE	11
Three query patterns.....	12
The browse pattern	12
The drill-down pattern.....	13
The invocation pattern.....	13
Inquiry API functions.....	14
find_binding.....	15
find_business	17
find_service.....	19
find_tModel.....	21
get_bindingDetail	22
get_businessDetail.....	23
get_businessDetailExt.....	24
get_serviceDetail	25
get_tModelDetail.....	26
Publishing API functions	27
Special considerations around categorization	28
delete_binding	28
delete_business	30
delete_service	31
delete_tModel.....	32
discard_authToken	34
get_authToken	35
get_registeredInfo.....	36
save_binding.....	37
save_business	39
save_service.....	42
save_tModel.....	43
APPENDIX A: ERROR CODE REFERENCE.....	45
Error Codes.....	45
dispositionReport overview.....	46

JUNE 28, 2002

PAGE 2

APPENDIX B: SOAP USAGE DETAILS.....	48
<i>Support for SOAPAction</i>	48
<i>Support for SOAP Actor</i>	48
<i>Support for SOAP encoding</i>	48
<i>Support for SOAP Fault</i>	48
<i>Support for SOAP Headers</i>	48
<i>Document encoding conventions – default namespace support</i>	48
<i>UTF-8 to Unicode: SOAP listener behavior</i>	49
APPENDIX C: XML USAGE DETAILS.....	50
<i>Use of multiple languages in the description elements</i>	50
<i>Valid Language Codes</i>	50
<i>Default Language Codes</i>	50
<i>ISSUE for Validation: XML namespace declaration</i>	50
<i>Support for XML Encoding</i>	51
APPENDIX D: SECURITY MODEL IN THE PUBLISHERS API.....	52
<i>Achieving wire level privacy: All methods are secured via SSL</i>	52
<i>Authentication</i>	52
<i>Establishing credentials</i>	52
<i>Authentication tokens are not portable</i>	52
<i>Generating Authentication Tokens</i>	52
<i>Per-account space limits</i>	53
APPENDIX E: SEARCH QUALIFIERS	54
<i>General form of search qualifiers</i>	54
<i>Search Qualifiers enumerated</i>	54
<i>Search Qualifier Precedence</i>	55
<i>Locale Details</i>	55
APPENDIX F: RESPONSE MESSAGE REFERENCE	56
APPENDIX G: REDIRECTION VIA HOSTINGREDIRECTOR ELEMENT	57
<i>Special situations requiring the hostingRedirector</i>	57
<i>Using the hostingRedirector data</i>	57
<i>Stepwise overview</i>	57
APPENDIX H: DETAILS ON THE VALIDATE_CATEGORIZATION CALL	59
<i>validate_categorization</i>	59
APPENDIX I: UTILITY TMODELS AND CONVENTIONS	61
<i>UDDI Type Taxonomy</i>	61
<i>UDDI Registry tModels</i>	63
<i>UDDI Core tModels - Taxonomies</i>	63
<i>UDDI Core tModels - Other</i>	64
<i>Registering tModels within the Type Taxonomy</i>	66
REFERENCES	67
CHANGE HISTORY	68

Copyright © 2000-2002 by Ariba, Inc., International Business Machines Corporation, Microsoft Corporation. All Rights Reserved.

These documents are provided by the companies named above ("Licensors") under the following license. By using and/or copying this document, or the document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the document from which this statement is linked, in any medium for any purpose and without fee or royalty under copyrights is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

1. A link to the original document.
2. An attribution statement: "Copyright © 2000-2002 by Ariba, Inc., International Business Machines Corporation, Microsoft Corporation. All Rights Reserved."

If the Licensors own any patents or patent applications which that may be required for implementing and using the specifications contained in the document in products that comply with the specifications, upon written request, a non-exclusive license under such patents shall be granted on reasonable and non-discriminatory terms.

THIS DOCUMENT IS PROVIDED "AS IS," AND LICENSORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

LICENSORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

Introduction

Document Overview

This document describes the programming interface that is exposed by all instances of the Universal Description, Discovery & Integration (UDDI) registry. The primary audience for this document is programmers that want to write software that will directly interact with a UDDI *Operator Site*¹.

What is this UDDI anyway?

UDDI is the name of a group of web-based registries that expose information about a business or other entity² and its technical interfaces (or API's). These registries are run by multiple *Operator Sites*, and can be used by any business that wants to make their information available, as well as anyone that wants to find that information. There is no charge for using the basic services of these operating sites.

By accessing any of the public UDDI *Operator Sites*, anyone can search for information about web services³ that are made available by or on behalf of a business. The benefit of having access to this information is to provide a mechanism that allows others to discover what technical programming interfaces are provided for interacting with a business for such purposes as electronic commerce, etc. The benefit to the individual business is increased exposure in an electronic commerce enabled world.

The information that a business can register includes several kinds of simple data that help others determine the answers to the questions "who, what, where and how". Simple information about a business – information such as name, business identifiers (D&B D-U-N-S Number®, etc.), and contact information answers the question "Who?". "What?" involves classification information that includes industry codes and product classifications, as well as descriptive information about the services that are available for electronic interchange. Answering the question "Where?" involves registering information about the URL or email address (or other address) through which each type of service is accessed⁴. Finally, the question "How?" is answered by registering references to information about specifications that describe how a particular software package or technical interface functions. These references are called tModels in the UDDI documentation.

Compatible registries

This programmer's reference, coupled with the UDDI API schema (uddiAPI.xsd), defines a programming interface that is available for free public use. Software developers, businesses and

¹ *Operator Site* is a term used to describe an implementation of this specification that participates in the public cloud of UDDI sites under special contract.

² The term *business* is used in a general sense to refer to an operating concern or any other type of organization throughout this document. This use is not intended to preclude other organizational forms.

³ *Web Service* is a term used to describe technical services that are exposed via some public standard. Examples include purchasing services, catalog services, search services, shipping or postal services exposed over transports like HTTP or electronic mail.

⁴ The information about the service point or address at which a service is exposed is sometimes referred to using the technical term *binding information*. design specs refer to this using the term *bindingTemplate*.

others are encouraged to define products and tools that make use of this API and to build registries that are compatible with the API defined in this specification.

What are tModels?

In order for two or more pieces of software to be compatible with each other – that is, compatible enough to be able to exchange data for the purpose of achieving a desirable result – they must share some design goals and specifications in common. The registry information model that each UDDI site supports is based on this notion of shared specifications.

In the past, to build compatible software, two companies only had to agree to use the same specification, and then test their software. However, with UDDI, companies need a way to publish information about the specifications and versions of specifications that were used to design their advertised services. To accommodate the need to distinctly identify public specifications (or even private specifications shared only with select partners), information about the specifications themselves needs to be discoverable. This information about specifications – a classic metadata construct – is called a tModel within UDDI.

The tModel concept serves a useful purpose in discovering information about services that are exposed for broad use. To get a clearer understanding, let's consider an example.

An example:

Suppose your business bought a software package that let you automatically accept electronic orders via your web site. Using one of the public UDDI sites, you could advertise the availability of this electronic commerce capability.

One of the reasons you chose this particular software package was its widespread popularity. In fact the salesperson that sold you the software made a point of highlighting a feature that gives your new software its broad appeal – the use and support of a widely used set of XML business documents to accommodate automatic business data interchange.

As you installed and configured your new software, this software automatically consulted one of the public UDDI sites and identified compatible business partners. It did this by looking up each business you identified, and located those that had already advertised support for electronic commerce services that are compatible with your own.

The configuration software accomplishes this by taking advantage of the fact that a tModel has been registered for a full specification, and in the service elements for each business, the tModel keys for this specification was referenced.


In general, it's pretty safe to think of the tModel keys within a service description as a fingerprint that can be used to trace the compatibility origins of a given service. Since many services will be constructed or programmed to be compatible with a given specification, references to information about specifications (by way of tModel entries and tModel references) don't have to be repeated with each registered electronic commerce service.

For programmers that write the software that will be used by businesses, tModels provide a common point of reference that allows compatible services to be easily identified. For businesses that use this software, the benefit is greatly reduced work in determining which particular services are compatible with the software you write. Finally, for software vendors and standards organizations, the ability to register information about a specification and then find implementations of web services that are compatible with a given tModel helps customers immediately realize the benefits of a widely used design.

Classification and Identification information

One of the immediate benefits of registering business information at one of the UDDI *Operator Instances* is the ability to specify one or more classification, or category codes for your business. Many such codes exist – NAICS, UN/SPC, SIC Codes, etc. – and are widely used to classify businesses, industries, and product categories. Other (and there are many) classifications designate geographic information, or membership in a given organization.

Each UDDI site provides a way to add any number of classifications to a business registration. This information allows simple searching to be done on the information contained in the public registries. More important, registering information such as industry codes, product codes, geography codes and business identification codes (such as D&B D-U-N-S Numbers®) allow other search services to use this core classification information as a starting point to provide added-value indexing and classification while still referencing your UDDI information.



Design & Architecture

The UDDI programmer's API is designed to provide a simple request/response mechanism that allows discovery of businesses, services and technical service binding information.

Design Principles

The primary principal guiding the design of this programmers API was simplicity. Care has been taken to avoid complexity, overlap, and also to provide direct access to the appropriate levels of registered information with a minimum of programming overhead and round tripping.

Security

Accessing UDDI programmatically is accomplished via API calls defined in this programmer's reference. Two types of APIs are defined. A publishers API is provided for interactions between programs and the UDDI registry for the purpose of storing or changing data in the registry. An inquiry API is provided for programs that want to access the registry to read information from the registry.

Authenticated access is required to use the publishers API. Each *Operator Site* is responsible for selecting and implementing an authentication protocol that is compatible with the publishers API, as well as providing a new user sign-up mechanism. Before using any of the publisher API functions, the caller is responsible for signing up with one or more *Operator Sites* or compatible registries and establishing user credentials.

The Inquiry API functions are exposed as SOAP messages over HTTP protocol. No authentication is required to make use of the Inquiry API functions.

Versioning

In any programmers API, as well as any message set, versioning issues arise as time passes. Changes to an API over time can result in requests being misunderstood or processed incorrectly unless one can determine whether the version of the API being provided matches the version of the API used by a requesting party.

In order to facilitate a proper and controlled version match, the entire API defined by this programmer's reference is version stamped. Since the API itself is based on XML messages transmitted in SOAP envelopes over HTTP⁵, this version stamp takes the form of an XML attribute.

All of the messages defined in this API must be transmitted with an accompanying application version attribute. This attribute is named "*generic*"⁶ and is present on all messages. Each time this specification is modified, an ensuing requirement is placed on all *Operator Sites* to support generic 1.0, the current generic and at least one prior generic, if any. Compatible registries are encouraged to support at a minimum this 1.0 version.

⁵ HTTP is used as a general term here. HTTPS is used exclusively for all of the calls defined in the publishers API.

⁶ Versioning of application behavior is accommodated via the *generic* attribute independently from the structures defined in the accompanying schema. In general, this form of versioning is preferable because it is easier to specify a new behavior against the same structures than to try and get data structure definitions to reflect business rules. Versioning the actual schema structures would present considerable technical difficulties after more than a small number of deployed applications existed.

SOAP Messaging

SOAP is a method for using Extensible Markup Language (XML) for use in message passing and remote procedure call (RPC) protocols. SOAP has been jointly defined and submitted to the World Wide Web consortium (W3C) for consideration as a standard web protocol.

SOAP is being used in conjunction with HTTP to provide a simple mechanism for passing XML messages to *Operator Sites* using a standard HTTP-POST protocol. Unless specified, all responses will be returned in the normal HTTP response document.

See the appendix on SOAP specific implementations for more information on the way that UDDI *Operator Sites* use the SOAP schema as an envelope mechanism for passing XML messages.

XML conventions

The programming interface for UDDI is based on Extensible Markup Language (XML). See the appendix (XML usage details) for more information on specific XML constructs and limitations used in the specification of the programmers interface.

Error Handling

The first line of error reporting is governed by the SOAP specification. SOAP fault reporting and fault codes will be returned for most invalid requests, or any request where the intent of the caller cannot be determined.

If any application level error occurs in processing a request message, a dispositionReport structure will be returned to the caller instead of a SOAP fault report. Disposition reports contain error information that includes descriptions and typed keys that can be used to determine the cause of the error. Refer to the appendix "Error Codes" for a general understanding of error codes. API specific interpretations of error codes are described following each API reference page.

Many of the API constructs defined in this document allow one or more of a given type of information to be passed. These API calls conceptually each represent a request on the part of the caller. The general error handling treatment is to detect errors in a request prior to processing the request. Any errors in the request detected will invalidate the entire request, and cause a dispositionReport to be generated within a SOAP Fault structure (see appendix A). In the case of an API call that involves passing multiples of a given structure, the dispositionReport will call out only the first detected error, and is not responsible for reporting multiple errors or reflecting intermediate "good" data.

White Space

Operator Sites and compatible implementations will store all data exactly as provided with one exception. Any leading or trailing white space characters will be removed from each field, element or attribute. White space characters include carriage returns, line feeds, spaces, and tabs.

XML Encoding

Despite its cross platform goals, XML still permits a broad degree of platform dependent ordering to seep into software. One of the key areas of seep has to do with the way that multiple language encoding is allowable in the XML specification. For the purpose of this specification and all UDDI *Operator Sites* consistency in handling of data is essential. For this reason, the default collation order for data registered within an *Operator Site* is binary. See appendix B for more information

related to the use of byte order marks and UTF-8 and the way the SOAP listeners convert all requests to Unicode prior to processing.

API Reference

This API reference is divided into n logical sections. Each section addresses a particular programming focus. The sections are arranged in order according to the most common uses, and within each section alphabetically.

Special values within API syntax examples are shown in italics. In most cases, the following reference applies to these values:

- *uuid_key*: Access keys within all UDDI defined data elements are represented as universal unique identifiers (these are sometimes called a GUID). The name of the element or attribute designates the particular key type that is required. These keys are always formatted according to DCE UUID conventions with the one exception being *tModelKey* values, which are prefixed with a URN qualifier in the format "uuid:" followed by the UUID value.
- *generic*: This special attribute is a required metadata element for all messages. It is used to designate the specification version used to format the SOAP message. In the 1.0 version of the specification, this value is required to be "1.0". Any other value passed can result in an *E_unsupported* error.
- *xmlns*: This special attribute is a required metadata element for all messages. Technically, it isn't an attribute, but is formally called a namespace qualifier. It is used to designate is a universal resource name (URN) value that is reserved for all references to the UDDI schema. In the 1.0 version of the specification, this value is required to be "urn:uddi-org:api".
- *findQualifiers*: This special element is found in the inquiry API functions that are used to search (e.g. *find_binding*, *find_business*, *find_service*, *find_tModel*). This argument is used to signal special behaviors to be used with searching. See the Search Qualifiers appendix for more information.
- *maxRows*: This special qualifier is found in the inquiry API functions that are used to search (e.g. *find_binding*, *find_business*, *find_service*, *find_tModel*). This argument is used to limit the number of results returned from a request. When an *Operator Site* or compatible instance returns data in response to a request that contains this limiting argument, the number of results will not exceed the integer value passed. If a result set is truncated as a result of applying this limit, the result will include the *truncated* attribute with a value of *true*.
- *truncated*: The truncated attribute indicates that a maximum number of possible results has been returned. The actual limit set for applying this treatment is *Operator Site* policy specific, but in general should be a sufficiently large number so as to not normally be an issue. No behaviors such as paging mechanisms are defined for retrieving more data after a truncated limit. The intent is to support the average query, but to allow *Operator Sites* the leeway required to be able to manage adequate performance.
- *categoryBag*: Searches can be performed based on a cross section of categories. Several categories are broadly supported by all *Operator Sites* and provide three categorization dimensions. These are industry type, product and service type, and geography. Searches involving category information can be combined to cross multiple

dimensions. For this reason, these searches are performed matching on ALL of the categories supplied. The net effect in generic 1 is the ability to use embedded category information as hints about how the registering party has categorized themselves, but not to provide a full third party categorization facility. This is the realm of portals and marketplaces, and may be enhanced in future generics.

- *identifierBag*: Searches involving identifiers are performed matching on any supplied identifier (e.g. D&B D-U-N-S Number®, etc) for any of the primary elements that have identifierBag elements. These searches allow broad identity matching by returning a match when any keyedReference set used to search identifiers matches a registered identifier.
- *tModelBag*: Searches that match a particular technical fingerprint use UUID values to search for bindingTemplates with matching tModelKey value sets. When used to search for web services (e.g. the data described by a bindingTemplate structure), the concept of tModel signatures allows for highly selective searches for specific combinations of keys. For instance, the existence of a web service that implements all of the parts of the UDDI specifications can be accomplished by searching for a combination of tModel key values that correspond to the full set of specifications (the UDDI specification, for instance, is divided into at least 5 different, separately deployable tModels). At the same time, limiting the number of tModelKey values passed in a search can perform broader searches that look for any web service that implements a specific sub-part of the full specification. All tModelKey values are always expressed using a Universal Resource Name (URN) format that starts with the characters "uuid:" followed by a formatted Universally Unique Identifier (UUID) consisting of an octet of Hexidecimal digits arranged in the common 12-4-4-8 format pattern.

In all cases, the XML structures, attributes and element names shown in the API examples are derived from the Message API schema. For a full understanding of structure contents, refer to this schema. It is suggested that tools that understand schemas be used to generate logic that populates the structures used to make the API calls against an *Operator Site*.

Three query patterns

The Inquiry API provides three forms of query that follow broadly used conventions. These two forms match the needs of two types of software that are traditionally used with registries.

The browse pattern

Software that allows people to explore and examine data – especially hierarchical data – requires browse capabilities. The browse pattern characteristically involves starting with some broad information, performing a search, finding general result sets and then selecting more specific information for drill-down.

The UDDI API specifications accommodate the browse pattern by way of the *find_xx* API calls. These calls form the search capabilities provided by the API and are matched with summary return messages that return overview information about the registered information that match the supplied search criteria.

A typical browse sequence might involve finding whether a particular business you know about has any information registered. This sequence would start with a call to *find_business*, perhaps passing the first few characters of the businesses name that you already know. This returns a businessList result. This result is overview information (keys, names and descriptions) of the businessEntity information that matched the search results returned by *find_business*.

If you spot the business you are looking for, you can drill into their *businessService* information, looking for particular service types (e.g. purchasing, shipping, etc) using the *find_service* API call. Similarly, if you know the technical fingerprint (tModel signature) of a particular product and want to see if the business you've chosen supports a compatible service interface, you can use *find_binding*.

The drill-down pattern

Once you have a key for one of the four main data types managed by a UDDI or compatible registry⁷, you can use that key to access the full registered details for a specific data instance. The current UDDI data types are *businessEntity*, *businessService*, *bindingTemplate* and *tModel*. You can access the full registered information for any of these structures by passing a relevant key type to one of the *get_xx* API calls.

Continuing the example from the previous section on browsing, one of the data items returned by all of the *find_xx* return sets is key information. In the case of the business we were interested in, the *businessKey* value returned within the contents of a *businessList* structure can be passed as an argument to *get_businessDetail*. The successful return to this message is a *businessDetail* message containing the full registered information for the entity whose key value was passed. This will be a full *businessEntity* structure.

The invocation pattern

In order to prepare an application to take advantage of a remote web service that is registered within the UDDI registry by other businesses or entities, you need to prepare that application to use the information found in the registry for the specific service being invoked. This type of cross business service call has traditionally been a task that is undertaken at development time. This will not necessarily change completely as a result of UDDI registry entries, but one significant problem can be managed if a particular invocation pattern is employed.

Data obtained from the UDDI registry about a *bindingTemplate* information set represents the instance specifics of a given remote service. The program should cache this information and use it to contact the service at the registered address. Tools have automated the tasks associated with caching (or hard coding) location information in previously popular remote procedure technologies. Problems arise however when a remote service is moved without any knowledge on the part of the callers. Moves occur for a variety of reasons, including server upgrades, disaster recovery, and service acquisition and business name changes.

When a call fails using cached information obtained from a UDDI registry, the proper behavior is to query the UDDI registry where the data was obtained for fresh *bindingTemplate* information. The proper call is *get_bindingDetails* passing the original *bindingKey* value. If the data returned is different from the cached information, the service invocation should automatically retry. If the result of this retry is successful, the new information should replace the cached information.

By using this pattern with web services, a business using a UDDI *Operator Site* can automate the recovery of a large number of partners without undue communication and coordination costs. For example, if a business has activated a disaster recovery site, most of the calls from partners will fail when they try to invoke services at the failed site. By updating the UDDI information with the new address for the service, partners who use the invocation pattern will automatically locate the new service information and recover without further administrative action.

⁷ Keys within UDDI compatible registries that are not *Operator Sites* are not synchronized with keys generated by *Operator Sites*. There is no key portability mechanism defined for crossing from a replicated operator site to a compatible registry that is not part of the replicated *Operator Cloud*.

Inquiry API functions

The messages in this section represent inquiries that anyone can make of any UDDI *Operator Site* at any time. These messages all behave synchronously and are required to be exposed via HTTP-POST only. Other synchronous or asynchronous mechanisms may be provided at the discretion of the individual UDDI *Operator Site* or UDDI compatible registry.

The publicly accessible queries are:

- **find_binding**: Used to locate specific bindings within a registered businessService. Returns a bindingDetail message.
- **find_business**: Used to locate information about one or more businesses. Returns a businessList message.
- **find_service**: Used to locate specific services within a registered businessEntity. Returns a serviceList message.
- **find_tModel**: Used to locate one or more tModel information structures. Returns a tModelList structure.
- **get_bindingDetail**: Used to get full bindingTemplate information suitable for making one or more service requests. Returns a bindingDetail message.
- **get_businessDetail**: Used to get the full businessEntity information for a one or more businesses. Returns a businessDetail message.
- **get_businessDetailExt**: Used to get extended businessEntity information. Returns a businessDetailExt message.
- **get_serviceDetail**: Used to get full details for a given set of registered businessService data. Returns a serviceDetail message.
- **get_tModelDetail**: Used to get full details for a given set of registered tModel data. Returns a tModelDetail message.

find_binding

The find_binding message returns a bindingDetail message that contains a *bindingTemplates* structure with zero or more bindingTemplate structures matching the criteria specified in the argument list.

Syntax:

```
<find_binding serviceKey="uuid_key" generic="1.0" [ maxRows="nn" ]  
  xmlns="urn:uddi-org:api" >  
  [<findQualifiers/>]  
  <tModelBag/>  
</find_binding>
```

Arguments:

- **maxRows:** This optional integer value allows the requesting program to limit the number of results returned.
- **serviceKey:** This *uuid_key* is used to specify a particular instance of a businessService element in the registered data. Only bindings in the specific businessService data identified by the serviceKey passed will be searched.
- **findQualifiers:** This collection of findQualifier elements can be used to alter the default behavior of search functionality. See the Search Qualifiers appendix for more information.
- **tModelBag:** This is a list of tModel *uuid_key* values that represent the technical fingerprint to locate in a bindingTemplate structure contained within the businessService instance specified by the serviceKey value. If more than one tModel key is specified in this structure, only bindingTemplate information that exactly matches all of the tModel keys specified will be returned (logical AND). The order of the keys in the tModel bag is not relevant. All tModelKey values begin with a uuid URN qualifier (e.g. "uuid:" followed by a known tModel UUID value).

Returns:

This function returns a bindingDetail message on success. In the event that no matches were located for the specified criteria, the bindingDetail structure returned in the response will be empty (e.g. contain no bindingTemplate data.)

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the response message will contain the *truncated* attribute with the value of this attribute set to *true*.

Searching using tModelBag will also return any bindingTemplate information that matches due to hostingRedirector references. The resolved bindingTemplate structure will be returned, even if that bindingTemplate is owned by a different businessService structure.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed:** signifies that the *uuid_key* value passed did not match with any known *serviceKey* key or *ttModel* key values. The error structure will signify which condition occurred first.
- **E_tooManyOptions:** signifies that more than one mutually exclusive argument was passed.
- **E_unsupported:** signifies that one of the *findQualifier* values passed was invalid.

find_business

The find_business message returns a businessList message that matches the conditions specified in the arguments.

Syntax:

```
<find_business generic="1.0" [ maxRows="nn" ] xmlns="urn:uddi-org:api" >
  [<findQualifiers/>]
  <name/> | <identifierBag/> | <categoryBag/> | <tModelBag/> | <discoveryURLs/>
</find_business>
```

Arguments: All arguments to this call listed are mutually exclusive except findQualifiers

- **maxRows:** This optional integer value allows the requesting program to limit the number of results returned.
- **findQualifiers:** This collection of findQualifier elements can be used to alter the default behavior of search functionality. See the Search Qualifiers appendix for more information.
- **name:** This string value is a partial name. The businessList return contains businessInfo structures for businesses whose name matches the value passed (leftmost match).
- **identifierBag:** This is a list of business identifier references. The returned businessList contains businessInfo structures matching any of the identifiers passed (logical OR).
- **categoryBag:** This is a list of category references. The returned businessList contains businessInfo structures matching all of the categories passed (logical AND).
- **tModelBag:** The registered businessEntity data contains bindingTemplates that in turn contain specific tModel references. The tModelBag argument lets you search for businesses that have bindings that are compatible with a specific tModel pattern. The returned businessList contains businessInfo structures that match all of the tModel keys passed (logical AND). tModelKey values must be formatted as URN qualified UUID values (e.g. prefixed with "uuid:")
- **discoveryURLs:** This is a list of URL's to be matched against the data associated with the discoveryURL's contents of registered businessEntity information. To search for URL without regard to useType attribute values, pass the useType component of the discoveryURL elements as empty attributes. If useType values are included, then the match will be made only on registered information that match both the useType and URL value. The returned businessList contains businessInfo structures matching any of the URL's passed (logical OR).

Returns:

This function returns a businessList on success. In the event that no matches were located for the specified criteria, a businessList structure with zero businessInfo structures is returned.

In the event of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the businessList will contain the *truncated* attribute with the value set to *true*.

Searching using tModelBag will also return any businessEntity that contains bindingTemplate information that matches due to hostingRedirector references. In other words, the businessEntity

that contains a bindingTemplate with a hostingRedirector value referencing a bindingTemplate that matches the tModel search requirements will be returned.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_nameTooLong:** signifies that the partial name value passed exceeds the maximum name length designated by the *Operator Site*.
- **E_tooManyOptions:** signifies that more than one search argument was passed.
- **E_unsupported:** signifies that one of the findQualifier values passed was invalid.

find_service

The find_service message returns a serviceList message that matches the conditions specified in the arguments.

Syntax:

```
<find_service businessKey="uuid_key" generic="1.0" [ maxRows="nn" ]  
  xmlns="urn:uddi-org:api" >  
  [<findQualifiers/>]  
  <name/> | <categoryBag/> | <tModelBag/>  
</find_service>
```

Arguments: The *name*, *categoryBag* and *tModelBag* arguments are mutually exclusive

- **maxRows:** This optional integer value allows the requesting program to limit the number of results returned.
- **businessKey:** This *uuid_key* is used to specify a particular BusinessEntity instance.
- **findQualifiers:** This collection of findQualifier elements can be used to alter the default behavior of search functionality. See the Search Qualifiers appendix for more information.
- **name:** This string value represents a partial name. Any businessService data contained in the specified businessEntity with a matching partial name value gets returned.
- **categoryBag:** This is a list of category references. The returned serviceList contains businessInfo structures matching all of the categories passed (logical AND).
- **tModelBag:** This is a list of tModel *uuid_key* values that represent the technical fingerprint to locate within a bindingTemplate structure contained within any businessService contained by the businessEntity specified. If more than one tModel key is specified in this structure, only businessServices that contain bindingTemplate information that matches all of the tModel keys specified will be returned (logical AND)

Returns:

This function returns a serviceList on success. In the event that no matches were located for the specified criteria, the serviceList structure returned will contain an empty businessServices structure. This signifies zero matches.

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the serviceList will contain the *truncated* attribute with the value of this attribute set to *true*.

Searching using tModelBag will return serviceInfo structure for all qualifying businessService data, including matches due to hostingRedirector references. In other words, if the businessEntity whose businessKey is passed as an argument contains a bindingTemplate with a hostingRedirector value, and that value references a bindingTemplate that matches the tModel search requirements, then the serviceInfo for the businessService containing the hostingRedirector will be returned.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed:** signifies that the *uuid_key* value passed did not match with any known businessKey key or tModel key values. The error structure will signify which condition occurred first.
- **E_nameTooLong:** signifies that the partial name value passed exceeds the maximum name length designated by the *Operator Site*.
- **E_tooManyOptions:** signifies that more than one mutually exclusive argument was passed.
- **E_unsupported:** signifies that one of the findQualifier values passed was invalid.

find_tModel

This find_tModel message is for locating a list of tModel entries that match a set of specific criteria. The response will be a list of abbreviated information about tModels that match the criteria (tModelList).

Syntax:

```
<find_tModel generic="1.0" [ maxRows="nn" ] xmlns="urn:uddi-org:api" >
  [<findQualifiers/>]
  <name/> | <identifierBag/> | <categoryBag/>
</find_tModel>
```

Arguments: The arguments to this call are mutually exclusive except findQualifiers

- **maxRows:** This optional integer value allows the requesting program to limit the number of results returned.
- **findQualifiers:** This collection of findQualifier elements can be used to alter the default behavior of search functionality. See the Search Qualifiers appendix for more information.
- **name:** This string value represents a partial name. The returned tModelList contains tModelInfo structures for businesses whose name matches the value passed (leftmost match).
- **IdentifierBag:** This is a list of business identifier references. The returned tModelList contains tModelInfo structures matching any of the identifiers passed (logical OR).
- **categoryBag:** This is a list of category references. The returned tModelList contains tModelInfo structures matching all of the categories passed (logical AND).

Returns:

This function returns a tModelList on success. In the event that no matches were located for the specified criteria, an empty tModelList structure will be returned (e.g. will contain zero tModelInfo structures). This signifies zero matches.

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the tModelList will contain the *truncated* attribute with the value of this attribute set to *true*.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_nameTooLong:** signifies that the partial name value passed exceeds the maximum name length designated by the *Operator Site*.
- **E_tooManyOptions:** signifies that more than one mutually exclusive argument was passed.
- **E_unsupported:** signifies that one of the findQualifier values passed was invalid.

get_bindingDetail

The get_bindingDetail message is for requesting the run-time bindingTemplate information location information for the purpose of invoking a registered business API.

Syntax:

```
<get_bindingDetail generic="1.0" xmlns="urn:uddi-org:api" >
  <bindingKey/>
  [ <bindingKey/> ...]
</get_bindingDetail>
```

Arguments:

- **bindingKey** : one or more *uuid_key* values that represent specific instances of known bindingTemplate data.

Behavior:

In general, it is recommended that bindingTemplate information be cached locally by applications so that repeated calls to a service described by a bindingTemplate can be made without having to make repeated round trips to an UDDI registry. In the event that a call made with cached data fails, the get_bindingDetail message can be used to get fresh bindingTemplate data. This is useful in cases such as a service you are using relocating to another server or being restored in a disaster recovery site.

Returns:

This function returns a bindingDetail message on successful match of one or more bindingKey values. If multiple bindingKey values were passed, the results will be returned in the same order as the keys passed.

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the bindingDetail result will contain the *truncated* attribute with the value of this attribute set to *true*.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known bindingKey key values. No partial results will be returned – if any bindingKey values passed are not valid bindingKey values, this error will be returned.

get_businessDetail

The get_businessDetail message returns complete businessEntity information for one or more specified businessEntities.

Syntax:

```
<get_businessDetail generic="1.0" xmlns="urn:uddi-org:api" >
  <businessKey/>
  [ <businessKey/> ...]
</get_businessDetail>
```

Arguments:

- **businessKey** : one or more *uuid_key* values that represent specific instances of known businessEntity data.

Returns:

This function returns a businessDetail message on successful match of one or more businessKey values. If multiple businessKey values were passed, the results will be returned in the same order as the keys passed.

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the businessDetail response message will contain the *truncated* attribute with the value of this attribute set to *true*.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey values. No partial results will be returned – if any businessKey values passed are not valid, this error will be returned.

get_businessDetailExt

The get_businessDetailExt message returns extended businessEntity information for one or more specified businessEntitys. This message returns exactly the same information as the get_businessDetail message, but may contain additional attributes if the source is an external registry (not an *Operator Site*) that is compatible with this API specification.

Syntax:

```
<get_businessDetailExt generic="1.0" xmlns="urn:uddi-org:api" >
  <businessKey/>
  [ <businessKey/> ...]
</get_businessDetailExt>
```

Arguments:

- **businessKey** : one or more *uuid_key* values that represent specific instances of known businessEntity data.

Returns:

This function returns a businessDetailExt message on successful match of one or more businessKey values. If multiple businessKey values were passed, the results will be returned in the same order as the keys passed.

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the businessDetailExt response message will contain the *truncated* attribute with the value of this attribute set to *true*.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey values. No partial results will be returned – if any businessKey values passed are not valid, this error will be returned.
- **E_unsupported**: signifies that the implementation queried does not support the extended detail function. If this occurs, businessDetail information should be queried via the get_businessDetail API. *Operator Sites* will not return this code, but will instead return a businessDetailExt result with full businessDetail information embedded.

get_serviceDetail

The get_serviceDetail message is used to request full information about a known businessService structure.

Syntax:

```
<get_serviceDetail generic="1.0" xmlns="urn:uddi-org:api" >
  <serviceKey/>
  [ <serviceKey/> ... ]
</get_serviceDetail>
```

Arguments:

- **serviceKey** : one or more *uuid_key* values that represent specific instances of known businessService data.

Returns:

This function returns a serviceDetail message on successful match of one or more serviceKey values. If multiple serviceKey values were passed, the results will be returned in the same order as the keys passed.

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the response will contain a *truncated* attribute with the value of this attribute set to *true*.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known serviceKey values. No partial results will be returned – if any serviceKey values passed are not valid, this error will be returned.

get_tModelDetail

The get_tModelDetail message is used to request full information about a known tModel structure.

Syntax:

```
<get_tModelDetail generic="1.0" xmlns="urn:uddi-org:api" >
  <tModelKey/>
  [ <tModelKey/> ...]
</get_tModelDetail>
```

Arguments:

- **tModelKey**: one or more URN qualified *uuid_key* values that represent specific instances of known tModel data. All tModelKey values begin with a uuid URN qualifier (e.g. "uuid:" followed by a known tModel UUID value.)

Returns:

This function returns a tModelDetail message on successful match of one or more tModelKey values. If multiple tModelKey values were passed, the results will be returned in the same order as the keys passed.

In the even of a large number of matches, an *Operator Site* may truncate the result set. If this occurs, the response will contain a *truncated* attribute with the value of this attribute set to *true*.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the URN qualified *uuid_key* values passed did not match with any known tModelKey values. No partial results will be returned – if any tModelKey values passed are not valid, this error will be returned. Any tModelKey values passed without a uuid URN qualifier will be considered invalid.
- **E_keyRetired**: signifies that the request cannot be satisfied because the owner has retired the tModel information. The tModel reference may still be valid and used as intended, but the information defining the tModel behind the key is unavailable.

Publishing API functions

The messages in this section represent inquiries that require authenticated⁸ access to an UDDI *Operator Site*. Each business should initially select one *Operator Site* to host their information. Once chosen, information can only be updated at the site originally selected.

The messages defined in this section all behave synchronously and are callable via HTTP-POST only. HTTPS is used exclusively for all of the calls defined in this publishers API.

The publishing API calls are:

- **delete_binding:** Used to remove an existing bindingTemplate from the bindingTemplates collection that is part of a specified businessService structure.
- **delete_business:** Used to delete registered businessEntity information from the registry.
- **delete_service:** Used to delete an existing businessService from the businessServices collection that is part of a specified businessEntity.
- **delete_tModel:** Used to delete registered information about a tModel. If there are any references to a tModel when this call is made, the tModel will be marked deleted instead of being physically removed.
- **discard_authToken:** Used to inform an *Operator Site* that a previously provided authentication token is no longer valid. See get_authToken.
- **get_authToken:** Used to request an authentication token from an *Operator Site*. Authentication tokens are required to use all other API's defined in the publishers API. This function serves as the programs equivalent of a login request.
- **get_registeredInfo:** Used to request an abbreviated synopsis of all information currently managed by a given individual.
- **save_binding:** Used to register new bindingTemplate information or update existing bindingTemplate information. Use this to control information about technical capabilities exposed by a registered business.
- **save_business:** Used to register new businessEntity information or update existing businessEntity information. Use this to control the overall information about the entire business. Of the save_x API's this one has the broadest effect.
- **save_service:** Used to register or update complete information about a businessService exposed by a specified businessEntity.
- **save_tModel:** Used to register or update complete information about a tModel.

⁸ Authentication is not regulated by this API specification. Individual *Operator Sites* will designate their own procedures for getting a userID and password.

Special considerations around categorization

Several of the API's defined in this section allow you to save categorization information that is used to support searches that use taxonomy references. These are currently the `save_business`, `save_service` and `save_tModel` APIs. Categorization is specified using an optional element named `categoryBag`, which contains namespace-qualified references to taxonomy keys and descriptions in `keyedReference` structures.

Data contained in the `keyValue` attribute of each `keyedReference` is validated against the taxonomy referenced by the associated `tModelKey`. Only valid `keyValues` will be stored as entered. *Operator Sites* may handle invalid `keyValues` by either fail the request or changing the `tModelKey` value to reference a non-validated "etc." taxonomy which accepts all `keyValues`. See Appendix H for details on the validation of taxonomic information. If the *Operator Site* chooses to fail categorization mechanisms, the error codes defined in appendix H will be passed back as the error code on the relevant API calls.

delete_binding

The `delete_binding` message causes one or more `bindingTemplate` to be deleted.

Syntax:

```
<delete_binding generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <bindingKey/>
  [ <bindingKey/> ...]
</delete_binding>
```

Arguments:

- **authInfo**: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the `get_authToken` API call.
- **bindingKey** : one or more `uuid_key` values that represent specific instances of known `bindingTemplate` data.

Returns:

Upon successful completion, a `dispositionReport` is returned with a single success indicator.

Caveats:

If any error occurs in processing this message, a `dispositionReport` structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the `uuid_key` values passed did not match with any known `bindingKey` values. No partial results will be returned – if any `bindingKey` values passed are not valid, this error will be returned.
- **E_authTokenExpired**: signifies that the authentication token value passed in the `authInfo` argument is no longer valid because the token has expired.

- **E_authTokenRequired:** signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch:** signifies that one or more of the bindingKey values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_operatorMismatch:** signifies that one or more of the bindingKey values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.

delete_business

The delete_business message is used to remove one or more businessEntity structures.

Syntax:

```
<delete_business generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <businessKey/>
  [ <businessKey/> ...]
</delete_business>
```

Arguments:

- **authInfo**: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- **businessKey** : one or more *uuid_key* values that represent specific instances of known businessEntity data.

Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey values. No partial results will be returned – if any businessKey values passed are not valid, this error will be returned.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that one or more of the businessKey values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_operatorMismatch**: signifies that one or more of the businessKey values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.

delete_service

The delete_service message is used to remove one or more businessService structures.

Syntax:

```
<delete_service generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <serviceKey/>
  [ <serviceKey/> ...]
</delete_service>
```

Arguments:

- **authInfo**: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- **serviceKey** : one or more *uuid_key* values that represent specific instances of known businessService data.

Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known serviceKey values. No partial results will be returned – if any serviceKey values passed are not valid, this error will be returned.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that one or more of the serviceKey values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_operatorMismatch**: signifies that one or more of the serviceKey values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.

delete_tModel

The delete_tModel message is used to remove or retire one or more tModel structures.

Syntax:

```
<delete_tModel generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <tModelKey/> [ <tModelKey/> ...]
</delete_tModel>
```

Arguments:

- **authInfo**: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- **tModelKey**: one or more URN qualified *uuid_key* values that represent specific instances of known tModel data. All tModelKey values begin with a uuid URN qualifier (e.g. "uuid:" followed by a known tModel UUID value.)

Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator.

Behavior:

If a tModel is deleted and any other managed data references to that tModel by *uuid_key* (e.g. within a categoryBag, identifierBag or within a tModelInstanceInfo structure) it will not be physically deleted as a result of this call. Instead it will be marked as hidden. Any tModels hidden in this way are still accessible to their owner, via the get_registeredInfo, but will be omitted from any results returned by calls to find_tModel. The details associated with a hidden tModel are still available to anyone that uses the get_tModelDetail message. Publishing parties that want to remove all details about a tModel from the system should call save_tModel, passing empty values in the data fields, before calling this function. A hidden tModel can be restored and made universally visible by invoking the save_tModel API at a later time, passing the key of the hidden tModel.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the URN qualified *uuid_key* values passed did not match with any known tModelKey values. No partial results will be returned – if any tModelKey values passed are not valid, this error will be returned. Any tModelKey values passed without a uuid URN qualifier will be considered invalid.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that one or more of the tModelKey values passed refers to data that is not controlled by the individual who is represented by the authentication token.

- **E_operatorMismatch:** signifies that one or more of the tModelKey values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.

discard_authToken

The discard_authToken message is used to inform an *Operator Site* that the authentication token can be discarded. Subsequent calls that use the same authToken may be rejected. This message is optional for *Operator Sites* that do not manage session state or that do not support the get_authToken message.

Syntax:

```
<discard_authToken generic="1.0" xmlns="urn:uddi-org:api" >  
  <authInfo/>  
</discard_authToken>
```

Arguments:

- **authInfo:** this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.

Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator. Discarding an expired authToken will be processed and reported as a success condition.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenRequired:** signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.

get_authToken

The get_authToken message is used to obtain an authentication token. Authentication tokens are opaque values that are required for all other publisher API calls. This message is not required for *Operator Sites* that have an external mechanism defined for users to get an authentication token. This API is provided for implementations that do not have some other method of obtaining an authentication token or certificate, or that choose to use userID and Password based authentication.

Syntax:

```
<get_authToken generic="1.0" xmlns="urn:uddi-org:api"
  userID="someLoginName"
  cred="someCredential"
/>
```

Arguments:

- **userID:** this required attribute argument is the user that an individual authorized user was assigned by an *Operator Site*. *Operator Sites* will each provide a way for individuals to obtain a UserID and password that will be valid only at the given *Operator Site*.
- **cred:** this required attribute argument is the password or credential that is associated with the user.

Returns:

This function returns an authToken message that contains a valid authInfo element that can be used in subsequent calls to publisher API calls that require an authInfo value.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_unknownUser:** signifies that the Operator Site that received the request does not recognize the userID and/or pwd argument values passed as valid credentials.

get_registeredInfo

The get_registeredInfo message is used to get an abbreviated list of all businessEntity keys and tModel keys that are controlled by the individual associated the credentials passed.

Syntax:

```
<get_registeredInfo generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
</get_registeredInfo>
```

Arguments:

- **authInfo:** this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.

Returns:

Upon successful completion, a registeredInfo structure will be returned, listing abbreviated business information in one or more businessInfo structures, and tModel information in one or more tModelInfo structures. This API is useful for determining the full extent of registered information controlled by a single user in a single call.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired:** signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired:** signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.

save_binding

The save_binding message is used to save or update a complete bindingTemplate structure. This message can be used to add or update one or more bindingTemplate structures to one or more existing businessService structures.

Syntax:

```
<save_binding generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <bindingTemplate/> [<bindingTemplate/>...]
</save_binding>
```

Arguments:

- **authInfo:** this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- **bindingTemplate:** one or more complete bindingTemplate structures. The order in which these are processed is not defined. To save a new bindingTemplate, pass a bindingTemplate structure with an empty bindingKey attribute value.

Behavior:

Each bindingTemplate structure passed must contain a serviceKey value that corresponds to a registered businessService controlled by the same person saving the bindingTemplate data. The net effect of this call is to establish the parent businessService relationship for each bindingTemplate affected by this call. If the same bindingTemplate (determined by matching bindingKey value) is listed more than once, any relationship to the containing businessService will be determined by processing order, which is determined by the position of the bindingTemplate data in first to last order.

Using this message it is possible to move an existing bindingTemplate structure from one businessService structure to another by simply specifying a different parent businessService relationship. Changing a parent relationship in this way will cause two businessService structures to be affected.

If a bindingTemplate being saved contains a hostingRedirector element, and that element references a bindingTemplate that itself contains a hostingRedirector element, an error condition (E_invalidKeyPassed) will be generated.

Returns:

This API returns a bindingDetail message containing the final results of the call that reflects the newly registered information for the effected bindingTemplate structures.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired:** signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.

- **E_authTokenRequired:** signifies that the authentication token value passed in the `authInfo` argument is either missing or is not valid.
- **E_keyRetired:** signifies that the request cannot be satisfied because one or more `uuid_key` values specified has previously been hidden or removed by the requester. This specifically applies to the `tModelKey` values passed.
- **E_invalidKeyPassed:** signifies that the request cannot be satisfied because one or more `uuid_key` values specified is not a valid key value, or that a `hostingRedirector` value references a `bindingTemplate` that itself contains a `hostingRedirector` value.
- **E_userMismatch:** signifies that one or more of the `uuid_key` values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_operatorMismatch:** signifies that one or more of the `uuid_key` values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.
- **E_accountLimitExceeded:** signifies that user account limits have been exceeded.

save_business

The `save_business` message is used to save or update information about a complete `businessEntity` structure. This API has the broadest scope of all of the `save_x` API calls in the publisher API, and can be used to make sweeping changes to the published information for one or more `businessEntity` structures controlled by an individual.

Syntax:

```
<save_business generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <businessEntity/> [<businessEntity/>...] | <uploadRegister/> [<uploadRegister/>...]
</save_business>
```

Arguments:

Only one type of `businessEntity` or `uploadRegister` arguments may be passed in a given `save_business` message. Any number of `businessEntity` or `uploadRegister` values can be passed in a single save (up to an *Operator Site* imposed policy limit), but the two types of parameters should not be mixed.

- **authInfo:** this required argument is an element that contains an authentication token. Authentication tokens are obtained using the `get_authToken` API call.
- **businessEntity:** one or more complete `businessEntity` structures can be passed. These structures can be obtained in advance by using the `get_businessDetail` API call or by any other means.
- **uploadRegister:** one or more resolvable HTTP URL addresses that each point to a single and valid `businessEntity` or `businessEntityExt` structure. This variant argument allows a registry to be updated to reflect the contents of an XML document that is URL addressable. The URL must return a pure XML document that only contains a `businessEntity` structure as its top-level element, and be accessible using the standard HTTP-GET protocol.

Behavior:

If any of the `uuid_key` values within in a `businessEntity` structure (e.g. any data with a key value regulated by a `businessKey`, `serviceKey`, `bindingKey`, or `tModelKey`) is passed with a blank value, this is a signal that the data that is so keyed is being inserted. This does not apply to structures that reference other keyed data, such as `tModelKey` references within `bindingTemplate` or `keyedReference` structures, since these are references.

To make this function perform an update to existing registered data, the keyed entities (`businessEntity`, `businessService`, `bindingTemplate` or `tModel`) should have `uuid_key` values that correspond to the registered data.

Data can be deleted with this function when registered information is different than the new information provided. One or more `businessService` and `bindingTemplate` structures that are found in the controlling *Operator Site* but are missing from the `businessEntity` information provided in or referenced by this call will be deleted from the registry after processing this call.

Data that is contained within one or more `businessEntity` can be rearranged with this function when data passed to this function redefines parent container relationships for other registered

information. For instance, if a new businessEntity is saved with information about a businessService that is registered already as part of a separate businessEntity, this will result in the businessService being moved from its current container to the new businessEntity. This only applies if the same party controls the data referenced.

If the uploadRegister URL method is used to save data, the *Operator Site* is required to make sure that the URL used is included in the discoveryURLs collection within the businessEntity structure. If the URL passed to do the upload is not contained in this collection, it will be added automatically with a useType value set to the type of structure (either businessEntity or businessEntityExt) found in the file used to perform the upload.

If the file located by the uploadRegister URL value is an extended business entity (businessEntityExt) structure, only the businessEntity data found within that structure will be registered.

If the businessEntity method is used to save data (e.g not via an uploadRegister URL reference), then the *Operator Site* will create a URL that is specific to the *Operator Site* that can be used to get (via HTTP-GET) the businessEntity structure being registered. This information will be added to (if not present already) the discoveryURLs collection automatically with a useType value of "businessEntity".

Returns:

This API returns a businessDetail message containing the final results of the call that reflects the new registered information for the businessEntity information provided.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired:** signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired:** signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_keyRetired:** signifies that the request cannot be satisfied because one or more *uuid_key* values specified has previously been hidden or removed by the requester. This specifically applies to the tModelKey values passed.
- **E_invalidKeyPassed:** signifies that the request cannot be satisfied because one or more *uuid_key* values specified is not a valid key value. This includes any tModelKey references that are unknown.
- **E_invalidURLPassed:** signifies that an error occurred with one of the uploadRegister URL values.
- **E_userMismatch:** signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_operatorMismatch:** signifies that one or more of the businessKey values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.

- **E_invalidCategory (20000):** signifies that the given keyValue did not correspond to a category within the taxonomy identified by a tModelKey value within one of the categoryBag elements provided.
- **E_categorizationNotAllowed (20100):** Restrictions have been placed by the taxonomy provider on the types of information that should be included at that location within a specific taxonomy. The validation routine chosen by the *Operator Site* has rejected this businessEntity for at least one specified category.
- **E_accountLimitExceeded:** signifies that user account limits have been exceeded.

save_service

The save_service message adds or updates one or more businessService structures.

Syntax:

```
<save_service generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <businessService/> [<businessService/>...]
</save_service>
```

Arguments:

- **authInfo:** this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- **businessService:** one or more complete businessService structures can be passed. These structures can be obtained in advance by using the get_serviceDetail API call or by any other means.

Behavior:

Each businessService structure passed must contain a businessKey value that corresponds to a registered businessEntity controlled by the same making the save_service request.. If the same businessService, or within these, bindingTemplate (determined by matching businessService or bindingKey value) is contained in more than one businessService argument, any relationship to the containing businessEntity will be determined by processing order – which is determined by first to last order of the information passed in the request. Using this message it is possible to move an existing bindingTemplate structure from one businessService structure to another, or move an existing businessService structure from one businessEntity to another by simply specifying a different parent businessEntity relationship. Changing a parent relationship in this way will cause two businessEntity structures to be affected.

Returns:

This API returns a serviceDetail message containing the final results of the call that reflects the newly registered information for the effected businessService structures.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired:** signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired:** signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_keyRetired:** signifies that the request cannot be satisfied because one or more *uuid_key* values specified has previously been hidden or removed by the requester. This specifically applies to the tModelKey values passed.

- **E_invalidKeyPassed:** signifies that the request cannot be satisfied because one or more *uuid_key* values specified is not a valid key value. This includes any *tModelKey* references that are unknown.
- **E_userMismatch:** signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_operatorMismatch:** signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.
- **E_invalidCategory (20000):** signifies that a *keyValue* did not correspond to a category within the taxonomy identified by the *tModelKey* in the *categoryBag* data provided.
- **E_categorizationNotAllowed (20100):** The taxonomy validation routine chosen by the *Operator Site* has rejected the *businessService* data provided.
- **E_accountLimitExceeded:** signifies that user account limits have been exceeded.

save_tModel

The *save_tModel* message adds or updates one or more *tModel* structures.

Syntax:

```
<save_tModel generic="1.0" xmlns="urn:uddi-org:api" >
  <authInfo/>
  <tModel/> [<tModel/>...] | <uploadRegister/> [<uploadRegister/>...]
</save_tModel>
```

Arguments:

- **authInfo:** this required argument is an element that contains an authentication token. Authentication tokens are obtained using the *get_authToken* API call.
- **tModel:** one or more complete *tModel* structures can be passed. If adding a new *tModel*, the *tModelKey* value should be passed as an empty element.
- **uploadRegister:** one or more resolvable HTTP URL addresses that each point to a single and valid *tModel* structure. This variant argument allows a registry to be updated to reflect the contents of an XML document that is URL addressable. The URL must return a pure XML document that only contains a *tModel* structure as its top-level element, and be accessible using the standard HTTP-GET protocol.

Behavior:

If any of the *uuid_key* values within in a *tModel* structure (e.g. *tModelKey*) is passed with a blank value, this is a signal that the data is being inserted.

To make this function perform an update to existing registered data, the *tModelKey* values should have *uuid_key* values that correspond to the registered data. All *tModelKey* values that are non-blank are formatted as urn values (e.g. the characters "uuid:" precede all UUID values for *tModelKey* values)

If a `tModelKey` value is passed that corresponds to a `tModel` that was previously hidden via the `delete_tModel` message, the result will be the restoration of the `tModel` to full visibility (e.g. available for return in `find_tModel` results again).

Returns:

This API returns a `tModelDetail` message containing the final results of the call that reflects the new registered information for the effected `tModel` structures.

Caveats:

If any error occurs in processing this message, a `dispositionReport` structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired:** signifies that the authentication token value passed in the `authInfo` argument is no longer valid because the token has expired.
- **E_authTokenRequired:** signifies that the authentication token value passed in the `authInfo` argument is either missing or is not valid.
- **E_keyRetired:** signifies that the request cannot be satisfied because one or more `uuid_key` values specified has previously been hidden or removed by the requester. This specifically applies to the `tModelKey` values passed.
- **E_invalidKeyPassed:** signifies that the request cannot be satisfied because one or more `uuid_key` values specified is not a valid key value. This will occur if a `uuid_key` value is passed in a `tModel` that does not match with any known `tModel` key.
- **E_invalidURLPassed:** an error occurred with one of the `uploadRegister` URL values.
- **E_userMismatch:** signifies that one or more of the `uuid_key` values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_operatorMismatch:** signifies that one or more of the `uuid_key` values passed refers to data that is not controlled by the *Operator Site* that received the request for processing.
- **E_invalidCategory:** signifies that the given `keyValue` did not correspond to a category within the taxonomy identified by a `tModelKey` value within one of the `categoryBag` elements provided.
- **E_categorizationNotAllowed:** Restrictions have been placed by the taxonomy provider on the types of information that should be included at that location within a specific taxonomy. The validation routine chosen by the *Operator Site* has rejected this `tModel` for at least one specified category.
- **E_accountLimitExceeded:** signifies that user account limits have been exceeded.

Appendix A: Error code reference

Error Codes

The following list of error codes can be returned in the `errno` values within a `dispositionReport` response to the API calls defined in this programmer's reference. The descriptions in this section are general and when used with the specific return information defined in the individual API call descriptions are useful for determining the reason for failures.

- **E_authTokenExpired:** (10110) signifies that the authentication token information has timed out.
- **E_authTokenRequired:** (10120) signifies that an invalid authentication token was passed to an API call that requires authentication.
- **E_accountLimitExceeded:** (10160) signifies that a save request exceeded the quantity limits for a given structure type. See "Structure Limits" in Appendix D for details.
- **E_busy:** (10400) signifies that the request cannot be processed at the current time.
- **E_categorizationNotAllowed:** (20100) Restrictions have been placed by the on the types of information that can be categorized within a specific taxonomy. The data provided does not conform to the restrictions placed on the category used. Used with categorization only.
- **E_fatalError:** (10500) signifies that a serious technical error has occurred while processing the request.
- **E_invalidKeyPassed:** (10210) signifies that the `uuid_key` value passed did not match with any known key values. The details on the invalid key will be included in the `dispositionReport` structure.
- **E_invalidCategory** (20000): signifies that the given `keyValue` did not correspond to a category within the taxonomy identified by the `tModelKey`. Used with categorization only.
- **E_invalidURLPassed:** (10220) signifies that an error occurred during processing of a save function involving accessing data from a remote URL. The details of the HTTP Get report will be included in the `dispositionReport` structure.
- **E_keyRetired:** (10310) signifies that a `uuid_key` value passed has been removed from the registry. While the key was once valid as an accessor, and is still possibly valid, the publisher has removed the information referenced by the `uuid_key` passed.
- **E_languageError:** (10060) signifies that an error was detected while processing elements that were annotated with `xml:lang` qualifiers. Presently, only the description element supports `xml:lang` qualifications.
- **E_nameTooLong:** (10020) signifies that the partial name value passed exceeds the maximum name length designated by the policy of an implementation or *Operator Site*.
- **E_operatorMismatch:** (10130) signifies that an attempt was made to use the publishing API to change data that is mastered at another *Operator Site*. This error is only relevant to the public *Operator Sites* and does not apply to other UDDI compatible registries.

- **E_success:** (0) Signifies no failure occurred. This return code is used with the `dispositionReport` for reporting results from requests with no natural response document.
- **E_tooManyOptions:** (10030) signifies that incompatible arguments were passed.
- **E_unrecognizedVersion:** (10040) signifies that the value of the *generic* attribute passed is unsupported by the *Operator Instance* being queried.
- **E_unknownUser:** (10150) signifies that the user ID and password pair passed in a `get_authToken` message is not known to the *Operator Site* or is not valid.
- **E_unsupported:** (10050) signifies that the implementer does not support a feature or API.
- **E_userMismatch:** (10140) signifies that an attempt was made to use the publishing API to change data that is controlled by another party. In certain cases, `E_operatorMismatch` takes precedence in reporting an error.

dispositionReport overview

Errors that are not reported by way of SOAP Faults are reported using the `dispositionReport` structure. This structure can be used to signal success for asynchronous requests as well.

Success Reporting with the dispositionReport structure

The general form of a success report is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
  <dispositionReport generic="1.0" operator="OperatorUrl"
    xmlns="urn:uddi-org:api" >
    <result errno="0" >
      <errInfo errCode="E_success" />
    </result>
  </dispositionReport>
</Body>
</Envelope>
```

Error reporting with the dispositionReport structure

All application errors are communicated via the use of the SOAP FAULT structure. The general form of an error report is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Body>
  <Fault>
    <faultcode>Client</faultcode>
    <faultstring>Client Error</faultstring>
```

```

<detail>
  <dispositionReport generic="1.0" operator="OperatorUrl"
    xmlns="urn:uddi-org:api" >
    <result errno="10050" >
      <errInfo errCode="E_notSupported">
        The findQualifier value passed is unrecognized.
      </errInfo>
    </result>
  </dispositionReport>
</detail>
</Fault>
</Body>
</Envelope>

```

Multiple *result* elements may be present within the dispositionReport structure, and can be used to provide very detailed error reports for multiple error conditions. The number of *result* elements returned within a disposition report is implementation specific. In general it is permissible to return an error response as soon as the first error in a request is detected.



Appendix B: SOAP usage details

This appendix covers the SOAP specific conventions and requirements for *Operator Sites*.

Support for SOAPAction

In version 1, the SOAPAction HTTP Header is required. The value passed in this HTTP Header must be an empty string that is surrounded by double quotes. Example:

```
POST /get_BindingDetail HTTP/1.1
Host: www.someOperator.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""
```

Support for SOAP Actor

In version 1 of the UDDI specification, the SOAP Actor feature is not supported. *Operator Sites* will reject any request that arrives with a SOAP Actor attribute.

Support for SOAP encoding

In version 1 of the UDDI specification, the SOAP encoding feature (section 5) is not supported. *Operator Sites* will reject any request that arrives with a SOAP encoding attribute.

Support for SOAP Fault

SOAP Fault applies when unknown API references invoked, etc. Applegate specific errors will be handled via the dispositionReport API within SOAP Fault structures (see appendix A). The following SOAP fault codes are used:

- **VersionMismatch:** An invalid namespace reference for the SOAP envelope element was passed. The valid namespace value is "http://www.xmlsoap.org/soap/envelope/".
- **MustUnderstand:** A SOAP header element was passed to an *Operator Site*. *Operator Sites* do not support any SOAP headers, and will return this error whenever a SOAP request is received that contains any Headers element.
- **Client:** A message was incorrectly formed or did not contain enough information to perform more exhaustive error reporting.

Support for SOAP Headers

In version 1 of the UDDI specification, SOAP Headers are not supported. *Operator Sites* are permitted to ignore any headers received. SOAP headers that have the must_understand attribute set to true will be rejected with a SOAP fault - MustUnderstand.

Document encoding conventions – default namespace support

Operator Sites are required to support the use of the default namespaces in SOAP request and response documents as shown in the following HTTP example:


```

POST /get_bindingDetail HTTP/1.1
Host: www.someoperator.org
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <get_bindingDetail generic="1.0"
      xmlns="urn:uddi-org:api">
      ...

```

UTF-8 to Unicode: SOAP listener behavior

The decision to use the UTF-8 encoding in all requests simplified the number of encoding variations that need to be handled within the XML interchanges used in this API specification. However, byte ordering and conversion issues can still arise. This section describes the behavior of the SOAP listeners that run at *Operator Sites* in regards to the way they convert data received into XML encoded in Unicode.

UTF-8 allows data to be transmitted with an optional three-position byte order mark (BOM) preceding the XML data. This BOM does not contain information that is useful for decoding the contents, but tells the receiving program the order that bytes within double-byte pairs occur within the data. Further analysis can then be performed to determine whether the XML received contains ASCII or Unicode characters. The BOM is not required to perform this analysis however, and it is safe for *Operator Sites* to remove the BOM prior to processing messages received.

Operator Sites must be prepared to accept messages that contain Byte Order Marks, but the BOM is not required to process SOAP messages successfully.

Data returned by all of the messages defined in this specification will not contain a BOM, and will be encoded as UTF-8 XML data.



Appendix C: XML Usage Details

This appendix explains the specifics of XML conventions employed across all UDDI *Operator Sites*. Implementations that desire to remain compliant with the behaviors of *Operator Sites* should follow these same conventions.

Use of multiple languages in the description elements.

Many of the messages defined in this programmers interface specification contain an element named *description*. Multiple descriptions are allowed to accommodate multiple language descriptions. These description elements are also permitted to be sent without an `xml:lang` attribute qualifier.

Only one description element is allowed to be passed to a `save_xx` API call without an `xml:lang` attribute qualifier. Elements passed in this way will be assigned the default language code of the registering party. This default language code is established at the time that publishing credentials are established with an individual *Operator Site*.

If more than one description element is sent in a document being stored, only the first description element in a particular structure may be sent without a `xml:lang` attribute qualifier. All subsequent description peers must contain an `xml:lang` qualifier.

Valid Language Codes

The valid values for language codes are to be specified as ISO language codes. Values for these codes and translation to other common language code sets can be found at:

<http://www.unicode.org/unicode/onlinedat/languages.html>

Only one description element is allowed for each language code used at any given container level.

Default Language Codes

A default ISO language code will be determined for a publisher at the time that a party establishes permissions to publish at a given *Operator Site* or implementation. This default language code will be applied to any description values that are provided with no language code.

On data returned via the SOAP interface, all descriptions will contain `xml:lang` qualifications.

ISSUE for Validation: XML namespace declaration

For use with the `xml:lang` language qualifiers, documents that contain this attribute will declare the `xml` namespace as:


`xmlns:xml="http://www.w3.org/1999/XMLSchema"`

This will occur at the top level of the instance document (in the `Envelope` element)

Support for XML Encoding

All messages to and from the Operator Site shall be encoded using the UTF-8 encoding, and all such messages shall have the 'encoding="UTF-8"' attribute on the initial line. Other encoding name variants, such as UTF8, UTF_8, etc. shall not be used. Therefore, to be explicit, the initial line shall be:

```
<?xml version="1.0" encoding="UTF-8" ?>
```



Appendix D: Security model in the publishers API

The Publishers API describes the messages that are used to control the content contained within an *Operator Site*, and can be used by compliant non-operator implementations that adhere to the behaviors described in this programmers reference specification.

Achieving wire level privacy: All methods are secured via SSL

All calls made to *Operator Sites* that use the messages defined in the publishers API will be transported using SSL encryption. *Operator Sites* will each provide a service description that exposes a bindingTemplate that makes use of HTTPS and SSL to secure the transmission of data.

Authentication

Each of the calls in the publishers API that change information at a given *Operator Site* requires the use of an opaque authentication token. These tokens are generated by or provided by each *Operator Site* independently, and are passed from the caller to the *Operator Site* in the element named *authInfo*.

These tokens are meaningful only to the *Operator Site* that provided them and are to be used according to the published policies of a given *Operator Site*.

Each party who has been granted publication access to a given *Operator Site* will be provided a token by the site. Obtaining this token is *Operator Site* specific.

Establishing credentials

Before any party can publish data within an *Operator Site*, credentials and permission to publish must be established with the individual operator. Generally, you will only need to interact with one *Operator Site* because all data published at any *Operator Site* is replicated automatically to all other *Operator Sites*. Establishing publishing credentials involves providing some verifiable identification information, contact information and establishing security credentials with the individual *Operator Site*. The specifics of these establishing credentials is *Operator Site* dependant, and all valid *Operator Sites* will provide a Web-based user interface via which to establish an identity and secure permissions to publish data.

Authentication tokens are not portable

Every registry implementation that adheres to these specifications will establish their own mechanism for token generation and authentication. The only requirement placed on token generation for use with the publishers API is that the tokens themselves must be valid string text that can be placed within the *authInfo* XML element. Given that binary to string translations are well understood and in common use, this requirement will not introduce hardships.

Authentication tokens are not required to be valid except at the *Operator Site* or implementation from which they originated. These tokens need only have meaning at a single *Operator Site* or implementation, and will not be expected to work across sites.

Generating Authentication Tokens

Many implementations are expected to require a login step. The *get_authToken* message is provided to accommodate those implementations that desire a login step. Security schemes that

are based on the convention of exchanging User ID and password credentials fall into this category. For implementations that desire this kind of security, the `get_authToken` API is provided as an optional means for generating a temporary authentication token.

Certificate based authentication and similar security mechanisms do not require this additional step of "logging in" and can directly pass compatible authentication token information (such as a certificate value) within the `authInfo` element provided on each of the publishers API messages. If certificate based authentication or similar security is employed by the choice of a given *Operator Site*, the use of the `get_authToken` and `discard_authToken` messages is optional.

Per-account space limits

Operator Sites may impose limits on the amount of data that can be published by a given user. The initial limits for a new user are:

- `businessEntity`: 1 per user account
- `businessService`: 4 per `businessEntity`
- `bindingTemplate`: 2 per `businessService`
- `tModel`: 10 per user account

Individual user accounts can negotiate per-account limits with the *Operator Site*.



Appendix E: Search Qualifiers

The inquiry API functions *find_binding*, *find_business*, *find_service*, and *find_tModel* each will accept an optional element named *findQualifiers*. This element argument is provided as a means to allow the caller to override default search behaviors.

General form of search qualifiers

The general form of the *findQualifiers* structure is:

```
<findQualifiers>
  <findQualifier>fixedQualifierValue</findQualifier>
  [<findQualifier>fixedQualifierValue</findQualifier> ...]
</findQualifiers>
```

Search Qualifiers enumerated

The value passed in each *findQualifier* element represents the behavior change desired by the caller. These values must come from the following list of qualifiers:

- **exactNameMatch**: signifies that leftmost name match behavior should be overridden. When this behavior is specified, only entries that exactly match the entry passed in the name argument will be returned.
- **caseSensitiveMatch**: signifies that the default case-insensitive behavior of a name match should be overridden. When this behavior is specified, case is relevant in the search results and only entries that match the case of the value passed in the name argument will be returned.
- **sortByNameAsc**: signifies that the result returned by a *find_x* or *get_x* inquiry call should be sorted on the name field in ascending alphabetic sort order. This sort is applied prior to any truncation of result sets. Only applicable on queries that return a *name* element in the topmost detail level of the result set. If no conflicting sort qualifier is specified, this is the default sort order for inquiries that return *name* values at this topmost detail level.
- **sortByNameDesc**: signifies that the result returned by a *find_x* or *get_x* inquiry call should be sorted on the name field in descending alphabetic sort order. This sort is applied prior to any truncation of result sets. Only applicable on queries that return a *name* element in the topmost detail level of the result set. This is the reverse of the default sort order for this kind of result.
- **sortByDateAsc**: signifies that the result returned by a *find_x* or *get_x* inquiry call should be sorted based on the date last updated in ascending chronological sort order (earliest returns first). If no conflicting sort qualifier is specified, this is the default sort order for all result sets. Sort qualifiers involving date are secondary in precedence to the *sortByName* qualifiers. This causes *sortByName* elements to be sorted within name by date, oldest to newest.
- **sortByDateDesc**: (default) signifies that the result returned by a *find_x* or *get_x* inquiry call should be sorted based on the date last updated in descending chronological sort order (most recent change returns first). Sort qualifiers involving date are secondary in precedence to the *sortByName* qualifiers. This causes *sortByName* elements to be sorted within name by date, oldest to newest.

At this time, these are the only qualifiers defined. *Operator Sites* may define more search qualifier values than these – but all *Operator Sites* and fully compatible software must support these qualifiers and behaviors.

Search Qualifier Precedence

Precedence of search qualifiers, when combined is as follows:

1. **exactNameMatch, caseSensitiveMatch:** These can be combined but are equal in precedence.
2. **sortByNameAsc, sortByNameDesc:** These are mutually exclusive, but equal in precedence.
3. **sortByDateAsc, sortByDateDesc:** These are mutually exclusive, but equal in precedence.

The precedence order is used to determine the proper ordering of results when multiple search qualifiers are combined.

Locale Details

The US English (EN_US) locale shall be used whenever a string comparison or alphabetic sort is specified. This applies to sortByNameAsc, sortByNameDesc, exactNameMatch.



Appendix F: Response message reference

Here we explain each of the response messages. These are technically defined in the UDDI API schema:

- **authToken:** This structure is return by the optional `get_authToken` message to return authentication information. The value returned is used in subsequent calls that require an `authInfo` value.
- **bindingDetail:** This structure is the technical information required to make a method call to an advertised web service. It is returned in response to the `get_bindingDetail` message.
- **businessDetail:** This structure contains full details for zero or more `businessEntity` structures. It is returned in response to a `get_businessDetail` message, and optionally in response to the `save_business` message.
- **businessDetailExt:** This structure allows UDDI compatible registries to define and share extended information about a `businessEntity`. *Operator Sites* support this message but return no additional data. This structure contains zero or more `businessEntityExt` structures. It is returned in response to a `get_businessDetailExt` message.
- **businessList:** This structure contains abbreviated information about registered `businessEntity` information. This message contains zero or more `businessInfo` structures. It is returned in response to a `find_business` message.
- **dispositionReport:** This structure is used to report the outcome of message processing and to report errors discovered during processing. This message contains one or more result structures. A special case – success – contains only one result structure with the special `erno` attribute value of `E_success (0)`.
- **registeredInfo:** This structure contains abbreviated information about all registered `businessEntity` and `tModel` information that are controlled by the party specified in the request. This message contains one or more `businessInfo` structures and zero or more `tModelInfo` structures. It is returned in response to a `get_registeredInfo` message.
- **serviceDetail:** This structure contains full details for zero or more `businessService` structures. It is returned in response to a `get_serviceDetail` message, and optionally in response to the `save_binding` and `save_service` messages.
- **serviceList:** This structure contains abbreviated information about registered `businessService` information. This message contains zero or more `serviceInfo` structures. It is returned in response to a `find_service` message.
- **tModelDetail:** This structure contains full details for zero or more `tModel` structures. It is returned in response to a `get_tModelDetail` message, and optionally in response to the `save_tModel` message.
- **tModelList:** This structure contains abbreviated information about registered `tModel` information. This message contains zero or more `tModelInfo` structures. It is returned in response to a `find_tModel` message.

Appendix G: redirection via hostingRedirector element

One of the main benefits of using a public *Operator Site* instance of an UDDI registry is to provide a single point of reference for determining the correct location to send a business service request to a remote web service. In general, the controller of a particular instance of bindingTemplate structure can be assured that by keeping the registered copy pointing to the proper server or invocation address, special conditions such as disaster recovery to a secondary site can be handled with a minimum of service disruption for customers or partners. The same holds true for those who choose to use a registry that is compatible with the UDDI API, but to a lesser degree.

In many cases, the API specified in the `get_bindingDetail` message is straightforward. Once a business or application knows of a service that needs to be invoked, the bindingTemplate information for this service can be cached until needed. In the event that the cached information fails at the time the partner web service is actually invoked (e.g. the `accessPoint` information in the cached bindingTemplate structure is used to invoke a remote partner service), the application can use the `bindingKey` in the cached information to get a fresh copy of the bindingTemplate information. This cached approach serves to prevent needless round trips to the registry.

Special situations requiring the hostingRedirector

Two special needs arise that cannot be directly supported by the *accessPoint* information in a bindingTemplate. These are:

- **Third Party Hosting of Technical Web Services:** A business chooses to expose a service that is actually hosted at a remote or third party site. Application Service Providers and Network Market Makers are common examples of this situation. In this situation, it is the actual third party that needs to control the actual value of the binding information.
- **Use specific access control to binding location:** In other situations, such as situation specific redirection based on the identity of the caller, or even time-of-day routing, it is necessary to provide the actual contact point information for the remote service in a more dynamic way than the cached `accessPoint` data would support.

For these cases, the bindingTemplate structures contain an alternative data element called *hostingRedirector*. The presence of a hostingRedirector element is mutually exclusive with the `accessPoint` information. This makes it possible to tell which method of gaining the actual bindingTemplate information that contains the `accessPoint` data to use.

Using the hostingRedirector data

When a bindingTemplate returned by UDDI registry contains a hostingRedirector element, the programmer uses this information to locate the actual bindingTemplate for the hosted service. The content of the hostingRedirector element is a bindingKey reference that refers to another bindingTemplate that contains the address of a redirector service that will respond to a `get_bindingDetail` message. The argument that gets passed to this message is the original bindingTemplate *uuid_key* value for the redirected service. The bindingTemplate returned by this redirected call must have a `accessPoint` element in it – this being the actual binding information for the redirected web service request

Stepwise overview

1. A business registers a bindingTemplate **A** for a remotely hosted or redirected business service **S**. This bindingTemplate contains a bindingKey value **Q** that references a second

bindingTemplate **B**. The bindingTemplate **B** is typically controlled by the organization that hosts the redirection service. This bindingTemplate **B** contains an accessPoint element that points to the actual hostingRedirector service **R**.

2. A program that wants to call the service **S** that is registered in step one gets the binding information for the advertised service. This bindingTemplate information contains a hostingRedirector element with the bindingKey **K** for the bindingTemplate **B**.
3. The programmer takes the bindingKey **K** for and issues a get_bindingDetail message against the UDDI registry that the original bindingTemplate **A** came from. This returns the data for bindingTemplate **B**. The programmer now has the address of the service that implements the redirection. This information is in the accessPoint element found in bindingTemplate **B**. This service, to be compliant, knows how to respond to a get_bindingDetail message.
4. Using the original binding key **Q** and issues a get_bindingDetails to the redirector service **R**. This service is responsible for returning the actual binding information for the redirected business service **S** or returning an error. The programmer has the choice of caching this bindingTemplate if desired.

Using this algorithm, an organization that hosts services for other businesses to use can control the information that is used to actually access the hosted service. This not only provides this hosting organization with the ability to manage situations such as disaster recovery locations, but also lets them specify the actual URL that is used to make a call to the actual business service. This URL can be keyed specifically to the caller, or can be a general location for the hosted or redirected service.

In any case, the original caller is able to find the technical web service (bindingTemplate) advertised within the actual business partner's data without having to know that any redirection occurred.

Appendix H: Details on the validate_categorization call

Whenever `save_business`, `save_service` or `save_tModel` are called, all contents of any included *categoryBag* information may be checked to see that it is properly coded to match existing categories. The reason given for this is to maximize consistency across category based searches.

Operator specific policy allows interpretation of various error returns to result in non-specified behavior. Consult the operations policy of the operator at which you register data to understand the specific behaviors of any validation performed.

validate_categorization

The `validate_categorization` service performs two functions. It is used to verify that a specific category (*keyValue*) exists within the given taxonomy. The service also optionally restricts the entities that may be classified within the category.

Syntax:

```
<validate_categorization generic="1.0" xmlns="name_qualifier" >
  <tModelKey/>
  <keyValue/>
  [ <businessEntity/> | <businessService/> | <tModel/> ]
</validate_categorization>
```

Arguments:

The optional `businessEntity`, `businessService` or `tModel` parameters are mutually exclusive.

- **tModelKey:** The identifier of a registered tModel that is used as a namespace qualifier that implies a specific taxonomy.
- **keyValue:** The *category identifier*⁹ of the category within the identified taxonomy.
- **businessEntity:** (optional) The `businessEntity` structure being categorized
- **businessService:** (optional) The business service structure being categorized.
- **tModel:** (optional) The tModel structure being categorized.

Behavior:

To validate categorizations of registry entries, it is sufficient to verify the category identified by the `keyValue` parameter exists within the taxonomy identified by the `tModelKey`.

Optionally, a `businessEntity`, `businessService`, or `tModel` may be passed with the required values. This additional information may be used by the taxonomy validation service to verify that the entity is properly classified within this taxonomy category.

⁹ A *category identifier* is the specific coded value that designates a category within taxonomy. An example would be NAICS code 247 with the appropriate category identifier being "247".

Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator.

Caveats:

If any error occurs in processing this message, a dispositionReport structure will be returned to the caller. The following error number information will be relevant:

- **E_invalidKeyPassed:** the tModelKey value passed didn't match any known tModel.
- **E_invalidCategory:** one of the keyValue values supplied did not correspond to a category within the taxonomy identified by the tModelKey.
- **E_categorizationNotAllowed:** The optional businessEntity, businessService, or tModel provided does not conform to restrictions placed on the given category by the taxonomy publisher.



Appendix I: Utility tModels and Conventions

In order to facilitate consistency in Service Description (tModel) registration, and provide a framework for their basic organization within the UDDI registry, a set of conventions has been established. This section describes the conventions for registration of Service Descriptions, as well as a set of Utility tModels that facilitate registration of common information and the services provided by the UDDI registry itself.

UDDI Type Taxonomy

The UDDI specifications provide a great deal of flexibility in terms of the types of information that may be registered. A type taxonomy has been established to assist in general categorization of the types of information registered. In this release, the type taxonomy has been developed for the categorization of Service Descriptions, or tModels. Business or Service types may be incorporated into this taxonomy at a later date.

The approach to categorization of tModels within the UDDI Type Taxonomy is consistent with that used for each of the other taxonomies. The categorization information for each tModel is added to the `<categoryBag>` elements in a `save_tModel` message. A `<keyedReference>` element is added to the category bag to indicate the type of tModel that is being registered.

The values used for keyed references are defined in the UDDI Type Taxonomy shown in the tModel description table below.

tModel Name: uddi-org:types
tModel Description: UDDI Type Taxonomy
tModel UUID: uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4

Taxonomy Values

The table below describes the UDDI types taxonomy. As the structure is hierarchical, the ParentID column indicates the parent-child relationships. The tModel key is the root of the structure. Categorization is allowed at all levels of the taxonomy, with the exception of the root key.

ID	ParentID	Allowed	Description
tModel	tModel	no	These types are used for tModels
identifier	tModel	yes	Unique identifier
namespace	tModel	yes	Namespace
categorization	tModel	yes	Categorization (taxonomy)
specification	tModel	yes	Specification for a Web Service
xmlSpec	specification	yes	Specification for a Web Service using XML messages

soapSpec	xmlSpec	yes	Specification for interaction with a Web Service using SOAP messages
wsdlSpec	specification	yes	Specification for a Web Service described in WSDL
protocol	tModel	yes	Protocol
transport	protocol	yes	Wire/transport protocol
signatureComponent	tModel	yes	Signature component

tModel: The UDDI type taxonomy is structured to allow for categorization of registry entries other than tModels. This key is the root of the branch of the taxonomy that is intended for use in categorization of tModels within the UDDI registry. Categorization is not allowed with this key.

identifier: An identifier tModel represents a specific set of values used to uniquely identify information. For example, a Dun & Bradstreet D-U-N-S® Number uniquely identifies companies globally. The D-U-N-S® Number taxonomy is an identifier taxonomy.

namespace: A namespace tModel represents a scoping constraint or domain for a set of information. In contrast to an identifier, a namespace does not have a predefined set of values within the domain, but acts to avoid collisions. It is similar to the namespace functionality used for XML.

categorization: A categorization tModel is used for information taxonomies within the UDDI registry. NAICS and UNSPSC are examples of categorization tModels.

specification: A specification tModel is used for tModels that define interactions with a Web Service. These interactions typically include the definition of the set of requests and responses or other types of interaction that are prescribed by the service. tModels describing XML, COM, Corba, or any other services are specification tModels.

xmlSpec: An xmlSpec tModel is a refinement of the specification tModel type. It is used to indicate that the interaction with the service is via XML. The UDDI API tModels are xmlSpec tModels.

soapSpec: Further refining the xmlSpec tModel type, a soapSpec is used to indicate that the interaction with the service is via SOAP. The UDDI API tModels are soapSpec tModels, in addition to xmlSpec tModels.

wsdlSpec: A tModel for a Web Service described using WSDL is categorized as a wsdlSpec.

protocol: A tModel describing a protocol of any sort.

transport: A transport tModel is a specific type of protocol. HTTP, FTP, and SMTP are types of transport tModels.

signatureComponent: A signature component is used to for cases where a single tModel can not represent a complete specification for a Web Service. This is the case for specifications like RosettaNet, where implementation requires the composition of three tModels to be complete - a general tModel indicating RNIF, one for the specific PIP, and one for the error handling services.

Each of these tModels would be of type signature component, in addition to any others as appropriate.

UDDI Registry tModels

The UDDI registry defines a number of tModels to define its core services. Each of the core tModels are listed in this section.

tModel Name: uddi-org:inquiry
tModel Description: UDDI Inquiry API - Core Specification
tModel UUID: uuid:4CD7E4BC-648B-426D-9936-443EAAC8AE23
Categorization: specification, xmlSpec, soapSpec

This tModel defines the inquiry API calls for interacting with the UDDI registry.

tModel Name: uddi-org:publication
tModel Description: UDDI Publication API - Core Specification
tModel UUID: uuid:64C756D1-3374-4E00-AE83-EE12E38FAE63
Categorization: specification, xmlSpec, soapSpec

This tModel defines the publication API calls for interacting with the UDDI registry.

tModel Name: uddi-org:taxonomy
tModel Description: UDDI Taxonomy API
tModel UUID: uuid:3FB66FB7-5FC3-462F-A351-C140D9BD8304
Categorization: specification, xmlSpec, soapSpec

This tModel defines the taxonomy maintenance API calls for interacting with the UDDI registry.

UDDI Core tModels - Taxonomies

An additional set of tModels has been established to assist in categorization within industry taxonomies. These tModels are described below.

tModel Name: ntis-gov:naics:1997
tModel Description: Business Taxonomy: NAICS (1997 Release)
tModel UUID: uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2

Categorization: categorization

This tModel defines the NAICS industry taxonomy.

tModel Name: unspsc-org:unspsc:3-1

tModel Description: Product Taxonomy: UNSPSC (Version 3.1)

tModel UUID: uuid:DB77450D-9FA8-45D4-A7BC-04411D14E384

Categorization: categorization

This tModel defines the UNSPSC product taxonomy.

tModel Name: uddi-org:misc-taxonomy

tModel Description: Other Taxonomy

tModel UUID: uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4

Categorization: categorization

This tModel defines an unidentified taxonomy.

UDDI Core tModels - Other

Additional tModels are defined to help register within leading industry encoding schemes and standard protocols. This list is expected to be expanded as appropriate as the UDDI business registry expands.

tModel Name: dnb-com:D-U-N-S

tModel Description: Dun & Bradstreet D-U-N-S® Number

tModel UUID: uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823

Categorization: identifier

This tModel is used for the Dun & Bradstreet D-U-N-S® Number identifier. Note that this tModel is initially registered as part of the UDDI core tModels. Once the registry is in production, management of this tModel is expected to be transferred to the Dun & Bradstreet publisher account. For more information, see <http://www.dnb.com>.

tModel Name: thomasregister-com:supplierID
tModel Description: Thomas Registry Suppliers
tModel UUID: uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039
Categorization: identifier

This tModel is used for the Thomas Register supplier identifier codes. Note that this tModel is initially registered as part of the UDDI core tModels. Once the registry is in production, custody of this tModel is expected to be transferred to the Thomas Register publisher account. For more information, see <http://www.thomasregister.com>.

tModel Name: uddi-org:smtp
tModel Description: E-mail based web service
tModel UUID: uuid:93335D49-3EFB-48A0-ACEA-EA102B60DDC6
Categorization: transport

This tModel is used to describe a web service that is invoked through SMTP email transmissions. These transmissions may be either between people or applications.

tModel Name: uddi-org:fax
tModel Description: Fax based web service
tModel UUID: uuid:1A2B00BE-6E2C-42F5-875B-56F32686E0E7
Categorization: protocol

This tModel is used to describe a web service that is invoked through fax transmissions. These transmissions may be either between people or applications.

tModel Name: uddi-org:ftp
tModel Description: File transfer protocol (ftp) based web service
tModel UUID: uuid:5FCF5CD0-629A-4C50-8B16-F94E9CF2A674
Categorization: transport

This tModel is used to describe a web service that is invoked through file transfers via the ftp protocol.

tModel Name: uddi-org:telephone
tModel Description: Telephone based web service
tModel UUID: uuid:38E12427-5536-4260-A6F9-B5B530E63A07
Categorization: specification

This tModel is used to describe a web service that is invoked through a telephone call and interaction by voice and/or touch-tone.

tModel Name: uddi-org:http
tModel Description: An http or web browser based web service
tModel UUID: uuid:68DE9E80-AD09-469D-8A37-088422BFBC36
Categorization: transport

This tModel is used to describe a web service that is invoked through a web browser and/or the http protocol.

Registering tModels within the Type Taxonomy

When a new tModel is registered within UDDI, its type can be classified within the framework of the UDDI Type Taxonomy. This classification provides additional hints to applications for what type of tModel is being registered. For each appropriate classification, a keyed reference is added to the categoryBag element for the tModel.

As an example, the Dun & Bradstreet D-U-N-S® Number is a type of identifier for an organization. Within the UDDI type taxonomy, the `dnb-com:D-U-N-S` tModel is classified as type identifier.

The categoryBag element of the tModel registered would be as follows :

```
<categoryBag>
  <keyedReference
    tModelKey = "uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
    keyValue = "identifier"
    keyName = "tModel is a unique identifier">
  </categoryBag>
```

tModelKey: This is the GUID for the UDDI Types taxonomy. It is required.

keyValue: This is the identifier for the categorization within the UDDI Types taxonomy. It is required.

keyName: This is the description of the identifier within the UDDI Types taxonomy. It is not required as a part of the registration, but simply provides additional information about the key selected.

References

This section contains URL pointers to various specifications and other documents that are pertinent in understanding this specification.

W3C specifications, notes and drafts

- XML 1.0
- XML Schema
- XML namespaces
- SOAP 1.1

UDDI specifications, white-papers and schemas

- UDDI API schema
- UDDI overview
- UDDI technical overview
- UDDI Operators specification
- UDDI.org
- UDDI XML structure reference

Change History

V1.00 30 September 2000 (Christopher Kurt). Added Appendix I, and updated table of contents as appropriate. Revised document date for final publication.

V1.01 27 March 2001 (Tom Glover). Corrected typographical errors.